MAANPUOLUSTUSKORKEAKOULU

VIRHEENKORJAUSALGORITMIT

Kandidaatintutkielma

Kadetti
Ville Parkkinen

99. kadettikurssi
Maasotalinja

Maaliskuu 2015

MAANPUOLUSTUSKORKEAKOULU

Kurssi	Linja
99. kadettikurssi	**Maasotalinja**
Tekijä **Kadetti Ville Parkkinen**	
Tutkielman nimi **VIRHEENKORJAUSALGORITMIT**	
Oppiaine johon työ liittyy Sotatekniikka	Säilytyspaikka MPKK:n kurssikirjasto
Aika Maaliskuu 2015	 Tekstisivuja 29, Liitesivuja 0

TIIVISTELMÄ

Virheenkorjausalgoritmit mahdollistavat kommunikaation häiriöisen kanavan välityksellä. Esimerkiksi reaaliaikaisessa kommunikaatiossa on suotavaa, ettei virheenkorjausta toteuteta uudelleenlähetyksin, vaan häiriöiden sietämiseksi viestiin lisätään redundanssia eli päällekkäisyyttä. Tätä kutsutaan kanavakoodaukseksi. Virheenkorjausalgoritmien suorituskykyä voidaan vertailla ainakin niiden etäisyydellä, hyvyydellä suhteessa kanavan kapasiteettiin sekä koodausvahvistuksella. Koodausvahvistus on näistä konkreettisin, joskaan se ei huomioi algoritmien informaatiosuhdetta. Tällä hetkellä suorituskykyisimpiä algoritmeja ovat konvoluutiokoodeihin kuuluvat turbokoodit, lohkokoodeihin kuuluvat LDPC-koodit sekä ketjukoodit, joissa käytetään ulompana koodina pitkän lohkokoon lohkokoodia ja sisempänä jotakin modernia konvoluutiokoodia.

AVAINSANAT

VIRHEENKORJAUSALGORITMIT, KANAVAKOODAUS, TIEDONSIIRTOKANAVAN KAPASITEETTI

SISÄLLYS

1 JOHDANTO

1.1 Aihealueen esittely

Useissa käytännön sovelluksissa on tarkoituksenmukaista kommunikoida siten, ettei viestin välittäminen perustu vastaanottajan kuittaukseen viestin perillemenosta ja uudelleenlähetykseen, mikäli tätä kuittausta ei saada. Varsinkin reaaliaikaisessa kommunikaatiossa on suotavaa, ettei uudelleenlähetyksiä tarvita. Sotilassovelluksissa uudelleenlähetykset saattavat myös lisätä altistusta vihollisen elektroniselle tiedustelulle sekä häirinnälle.

Kuittauksiin perustumattomassa kommunikaatiossa jokainen yksittäinen viesti pyritään välittämään onnistuneesti ensimmäisellä lähetyskerralla. Kommunikaatiokanavan häiriöiden sietämiseksi viestiin lisätään *redundanssia*, päällekkäisyyttä, jotta viestiin kertyneet virheet voitaisiin löytää ja korjata vastaanottopäässä. Näitä menetelmiä kutsutaan *virheenkorjausalgoritmeiksi* (engl. forward error correction, FEC) tai -koodeiksi, ja niiden käyttämistä kanavakoodaukseksi.

1.2 Tutkielman tavoitteet ja rakenne

Tämä työ tarjoaa lukijalle katsauksen FEC-algoritmien teoriaan, tärkeimpään käsitteistöön sekä kehitysnäkymiin. Sen tarkoitus on avata alan tärkeimpiä tuloksia ja käsitteitä perustellen ja esimerkein, olettamatta lukijalta kuitenkaan syvällistä matematiikan tuntemusta. Työn laajuus huomioiden, tarkoituksena ei ole esitellä uusia tuloksia.

Tutkimuskysymykset on aseteltu ja niiden käsittely jaettu lukuihin seuraavasti:

- Mitä mittareita FEC-algoritmien suorituskykyjen vertailuun käytetään? Mitä ominaisuuksia näillä mittareilla on ja voidaanko jotain tai joitain mittareita suositella erityisesti?
- Mitkä ovat FEC-algoritmien pääasialliset luokat ja näiden luokkien erityispiirteet? Minkätyyppiset FEC-algoritmit ovat tällä hetkellä suorituskykyisimpiä?

Luku 2 käsittelee edellisiä kysymyksiä ja luku 3 jälkimmäisiä. Lähdeluettelo sisältää alan merkittäviä artikkeleita sekä muuta hyväksi todettua kirjallisuutta, joiden avulla aiheeseen voi syventyä edelleen.

1.3 Aiemmat tutkimukset ja tutkielman rajaus

FEC-algoritmeja käytetään tiedonsiirron lisäksi muun muassa tiedon tallentamiseen. Tässä tutkielmassa niitä käsitellään vain edellisestä näkökulmasta, vaikka suurimmilta osin tällä kontekstilla ei olekaan merkitystä esitettyjen tulosten kannalta. Tiedonsiirrosta tutkielma on rajattu käsittelemään vain virheenkorjausta, huomioiden modulaation siinä määrin kuin se on tämän kannalta tarpeellista. Näiden lisäksi myös esimerkiksi kompressointi ja salaaminen ovat tiedonsiirrossa olennaisia tekijöitä, mutta niitä ei voida käsitellä tämän työn puitteissa.

Tutkielman matemaattisen sisällön osalta lukijan oletetaan hallitsevan lineaarialgebran sekä todennäköisyyslaskennan perusteet, mutta oleellisimmilta osiltaan se on pyritty muotoilemaan siten, ettei matemaattinen osaaminen ole välttämätöntä.

2 INFORMAATIOTEORIAA

Oletetaan, että haluamme välittää viestin kahden paikan välillä käyttäen näiden välistä *kanavaa*, joka ei kuitenkaan ole täysin häiriötön. Kanava voi olla esimerkiksi radiolinkki, valokaapeli tai analoginen puhelinyhteys – tällä ei ole teoreettisen tarkastelun kannalta merkitystä. Haluamme välittää viestin ensimmäisellä yrityksellä siten, että se vastaanotetaan sellaisena kuin sen lähetimme, siis toisin sanoen tavoittelemme virheetöntä kommunikointia epätäydellisen kanavan välityksellä. Yksi osa ratkaisua voisi olla kanavan parantaminen fyysisesti, esimerkiksi lähetystehoa nostamalla tai häiriötekijöitä eliminoimalla, mutta tässä työssä keskitymme systeemiseen lähestymistapaan: liitämme lähettimeen *kooderin* ja vastaanottimeen *dekooderin*. Näistä ensimmäinen lisää signaaliin *redundanssia*, päällekkäisyyttä, ja jälkimmäinen tunnistaa ja korjaa viestiin tulleet virheet tätä hyödyntäen.

Fyysiseen ratkaisuun verrattuna lähestymistapamme hintana on kooderin ja dekooderin monimutkaisuus, tästä seuraava vaatimus prosessorille sekä koodauksen aiheuttama lisäviive. Sitä voidaan kuitenkin käyttää myös fyysisen ratkaisun rinnalla sekä tilanteissa, joissa lähetystehoa ei voida nostaa tai häiriölähteisiin vaikuttaa. Voidaksemme seuraavassa luvussa perehtyä erilaisiin FEC-algoritmeihin, siis tapoihin toteuttaa systeeminen ratkaisumme, on ensin tutustuttava informaatioteoriaan ja koodausteoriaan. Nämä tutkivat lähestymistapamme kannalta olennaisia kysymyksiä “miten hyvän suorituskyvyn voimme saavuttaa”, sekä “miten löydämme hyviä virheenkorjauskoodeja”. Tässä luvussa tutustumme näiden peruskäsitteisiin ja -tuloksiin.

2.1 Kanavat

Ennen kuin voimme käsitellä virheiden korjausta tarkemmin, on määriteltävä täsmällisesti miten ne syntyvät. Tarvitsemme siis kanavalle matemaattisen mallin. Tämän tutkielman tarpeisiin riittävät seuraavaksi määriteltävät kaksi binäärisen kanavan mallia ja yksi analogisen kanavan malli. Kuten kaikki matemaattiset mallit, ovat nämäkin vain todellisuuden approksimaatioita. Ne ovat silti käyttökelpoisia ja parhaimmillaan hyvinkin tarkkoja, kuten esimerkiksi tutkimuksissa [5, 8].

Määritelmä 2.1. *Binäärinen symmetrinen kanava.* $\mathscr{A}_x = \mathscr{A}_y = \{0,1\}$,

$$P(y=0\,|\,x=0) = 1-f \qquad P(y=0\,|\,x=1) = f$$
$$P(y=1\,|\,x=0) = f \qquad P(y=1\,|\,x=1) = 1-f.$$

Tässä joukot $\mathscr{A}_x$ ja $\mathscr{A}_y$ määrittävät lähtevien ja vastaanotettujen viestien mahdolliset symbolit. Ehdolliset todennäköisyydet määrittävät virheiden esiintymisen. Esimerkiksi todennäköisyys $P(y = 1\,|\,x = 0) = f$ tarkoittaa, että jokainen lähetetty symboli '0' muuttuu kanavassa symboliksi '1' todennäköisyydellä f. Binäärinen symmetrinen kanava sopii malliksi silloin, kun binäärinen kanava tuottaa virheitä toisistaan riippumattomasti ja likimain samalla todennäköisyydellä symbolista riippumatta.

Esimerkki 2.2. Lähetetään viesti

0 1 0 0 1 1 0 0 0 1 1 1 0 0 0 0 1 1 1 1

binäärisen symmetrisen kanavan läpi, jonka virhetodennäköisyys $f = 0{,}2$. Viestissä on 20 symbolia, joten siihen voi odottaa tulevan kanavassa keskimäärin 4 virhettä. Vastaanotettu viesti voisi näyttää siis esimerkiksi tältä:

0 1 1 0 1 1 0 1 0 1 1 1 0 0 0 1 1 1 0 1.

Virhetodennäköisyys f ei ole sama asia kuin kanavassa häviävä informaatio. Vaikka emme olekaan määritelleet informaation käsitettä vielä tarkemmin, lienee selvää, että jos $f = 0$, viesti säilyy muuttumattomana ja kaikki informaatio säilyy. Edelleen jos $f = 1$ ja kaikki symbolit vaihtuvat niin kaikki informaatio säilyy, sillä vastaanottajan tarvitsee vain kääntää symbolit uudestaan ja hän voi olla varma, että viesti on virheetön. Pahimmassa tilanteessa $f = 0{,}5$, sillä tällöin jokainen vastaanotettu symboli on yhtä suurella todennäköisyydellä alkuperäinen tai virheellinen. Tässä tilanteessa yhtään informaatiota ei vastaanoteta.

Määritelmä 2.3. *Binäärinen purskevirhekanava.* $\mathscr{A}_x = \mathscr{A}_y = \{0, 1\}$. Kanava on joko normaalitilassa tai virhetilassa. Normaalitilassa se toimii virheettömästi, mutta siirtyy jokaisen symbolin viestittämisen jälkeen virhetilaan todennäköisyydellä p_1. Virhetilassa kanava toimii kuten binäärinen symmetrinen kanava (virhetodennäköisyydellä f), ja siirtyy jokaisen viestitetyn symbolin jälkeen takaisin virheettömään tilaan todennäköisyydellä p_2.

Esimerkki 2.4. Lähetetään esimerkin 2.2 viesti binäärisen purskevirhekanavan läpi, jonka siirtymätodennäköisyydet ovat $p_1 = p_2 = 0{,}2$ ja virhetilan virhetodennäköisyys $f = 0{,}5$. Alkuperäinen viesti ja mahdollinen esimerkki vastaanotetusta viestistä ovat:

0 1 0 0 1 1 0 0 0 1 1 1 0 0 0 0 1 1 1 1
0 1 0 0 1 0 0 1 0 0 0 1 0 0 0 0 1 1 1 1.

Binäärinen purskevirhekanava kuvaa sikäli paremmin useaa todellista tilannetta, että sen tuottamat virheet eivät ole toisistaan riippumattomia, vaan keskittyvät *virhepurskeiksi*. On huomattava, että vastaanottaja ei voi suoraan tietää, missä osassa vastaanotetussa viestissä kanava on ollut missäkin tilassa. Kanava on erikoistapaus *Gilbert-Elliot* -mallista [9, 6], joka on myös normaalitilassaan binäärinen symmetrinen kanava, jolloin ero kahden tilan välillä on vain virhetodennäköisyyksien suuruudessa.

Binäärinen purskevirhekanava voidaan muokata vastaamaan binääristä symmetristä kanavaa järjestelemällä lähetettävät bitit likimain satunnaiseen järjestykseen ja purkamalla tämä järjestys vastaanottopäässä. Tätä kutsutaan *lomitteluksi*. Sillä on kuitenkin hintansa, nimittäin teoriassa parempaan suorituskykyyn virheiden korjauksessa voidaan päästä algoritmilla joka koittaa tunnistaa purskevirheisen kanavan tilan [15, s. 186–190]. Yksinkertaisempien, tässä työssä käsiteltyjen algoritmien kohdalla lomittelun hintana on kuitenkin vain sen prosessointiin kuluva aika.

Määritelmä 2.5. *Additiivinen valkoinen kohina (engl. Additive White Gaussian Noise, AWGN).*

$$x_i \in \mathbb{R} \quad \forall i \in \{1, \ldots, k\}, \tag{2.1}$$

$$Y_i \sim N(x_i, n), \tag{2.2}$$

$$\frac{1}{n} \sum_{i=1}^{k} x_i^2 \leq P, \tag{2.3}$$

missä x_i on signaali ajanhetkellä i, k niiden lukumäärä, N normaalijakauma ja n mallin varianssi eli kohinan suuruus. Tehorajoite $P > 0$ kuvaa kanavan käytettävissä olevaa maksimitehoa.

Tämä on yksinkertainen analogisen kanavan malli, jossa signaalin oletetaan kulkevan vapaasti lähettimestä vastaanottimeen ilman monitie-etenemistä. Sen huomioon ottamiseksi voitaisiin käyttää esimerkiksi Rayleigh- tai Rice-kanavaa [11, s. 64–98], mutta AWGN-kanava riittää tämän työn tarpeisiin.

Tutustuttuamme lyhyesti kahteen esimerkkiin teoreettisista malleista virheiden syntymiselle, voimme seuraavaksi tarkastella niiden korjaamista. Kanavamalleja tarvitaan jatkossa aina laskettaessa virheenkorjausalgoritmien tehokkuuksia virheenkorjauksessa.

2.2 Koodin etäisyys

Tässä alaluvussa tutustumme kahteen yksinkertaiseen FEC-algoritmiin ja tarkastelemme niitä yksinkertaisen mutta yleisesti käytetyn mittarin, koodin *etäisyyden* valossa. Alamme siis lähestyä luvun 2 varsinaista tavoitetta, työkalujen löytämistä erilaisten korjausalgoritmien suorituskykyjen vertailemiseen.

Aloitamme maailman ensimmäisestä ei-triviaalista FEC-algoritmista, Hamming(7, 4)-algoritmista [12]. Seuraava määritelmä kuvaa sekä kooderin että dekooderin toiminnan lähetettäessä binäärinen, siis kahden symbolin aakkostoa käyttävä, viesti epätäydellisen kanavan läpi. Oletamme viestin pituuden olevan neljällä jaollinen ja mikäli näin ei ole, voimme aina lisätä viestin loppuun 1-3 symbolia ’0’, minkä jälkeen se täyttää vaatimuksen.

Määritelmä 2.6. *Hamming(7,4)*-algoritmi. Kooderi jakaa lähetteen neljän bitin mittaisiin *viesteihin* ja, aloittaen alkupäästä, lähettää kanavaan jokaista viestiä kohti taulukon 1 mukaisen seitsemän bitin *lohkon* eli *koodisanan*.

Taulukko 1: Hamming(7,4) -algoritmin viestit ja koodisanat

rivi	viesti	koodisana		rivi	viesti	koodisana
1	0000	0000000		9	1000	1000101
2	0001	0001011		10	1001	1001110
3	0010	0010111		11	1010	1010010
4	0011	0011100		12	1011	1011001
5	0100	0100110		13	1100	1100011
6	0101	0101101		14	1101	1101000
7	0110	0110001		15	1110	1110100
8	0111	0111010		16	1111	1111111

Vastaanottopäässä dekooderi kokoaa lähetteen tulkitsemalla jokaisen vastaanotetun koodisanan joko suoraan taulukon mukaan tai, mikäli vastaanotettua koodisanaa ei löydy taulukosta, niin sen koodisanan mukaan joka vastaanotetusta saadaan vaihtamalla yksi bitti.

Hamming-koodi voitaisiin määritellä myös esimerkiksi kääntämällä (ykköset nolliksi ja päinvastoin) määritelmän 2.6 jokaisen koodisanan tarkistusbitit, eli kolme viimeistä bittiä, ja sijoittamalla ne tämän jälkeen koodisanojen perästä muihin paikkoihin, niiden keskelle. Näin voitaisiin välttää koodisanoissa esiintyvää toistoa, erityisesti määritelmässä 2.6 esiintyvät koodisanat `0000000` ja `1111111` voivat käytännössä olla epätoivottavia.

Esimerkki 2.7. Lähetetään bitit `01101101` binäärisen symmetrisen kanavan ($f = 0{,}1$) läpi käyttäen Hamming(7,4)-algoritmia. Kooderi jakaa ensin lähetteen viesteihin `0110` ja `1101`, katsoo sitten taulukosta näitä vastaavan koodisanat `0110001` sekä `1101000` ja lähettää nämä kanavan läpi. Lähetetyt bitit ovat siis `01100011101000` ja vastaanotetut esimerkiksi `01100011101010`. Vastaanottopäässä dekooderi jakaa nämä koodisanoihin `0110001` ja `1101010`. Taulukon mukaan ensimmäinen näistä on suoraan rivillä 7, joten lähetteen ensimmäinen osa on `0110`, mutta jälkimmäistä ei löydy taulukosta. Kanavassa on siis tapahtunut ainakin yksi virhe. Rivin 14 koodisana poikkeaa vastaanotetusta vain toiseksi viimeisen bitin osalta, joten dekooderi tulkitsee lähetteen jälkimmäiseksi osaksi `1101`. Dekoodattu lähete on siis `01101101`, jotka ovat täsmälleen lähetetyt bitit.

Hamming(7,4)-algoritmin käytännöllisyys piilee siinä, että kaikki koodisanat eroavat toisistaan vähintään kolmella bitillä. Näin ollen mihin tahansa koodisanaan voi tulla kanavassa yksi virhe halutun dekoodauksen kärsimättä, sillä koodisana poikkeaa tällöin alkuperäisestä vain yhdessä kohdassa ja kaikista muista vähintään kahdessa. Toisaalta jos yksittäiseen koodisanaan tulee kanavassa virhe useampaan kuin yhteen bittiin, se ei missään tapauksessa dekoodaannu enää oikein. Tämä yksittäisen koodisanan virheellinen tulkinta, *lohkovirhe*, tapahtuu esimerkiksi binäärisessä symmetrisessä kanavassa todennäköisyydellä $1 - \left((1-f)^7 + 7f(1-f)^6\right)$. Esimerkissä 2.7 häiriösuhde $f = 0{,}1$, jolloin lohkovirheen todennäköisyys on $p_{LV} \approx 0{,}15$.

Voidaksemme todeta Hamming(7,4)-algoritmin virheenkorjauskyvyn konkreettisemmin, laskemme vielä todennäköisyyden sille, että yksittäinen vastaanotettu bitti eroaa vastaavasta lähetetystä bitistä. Tämä *bittivirhesuhde* (engl. bit error rate, BER) on algoritmillemme $\text{BER} = \frac{3}{7} p_{LV}$ [15, s. 17–18]. Jatkaen siis esimerkkiä 2.7, voimme vähentää Hamming(7,4)-algoritmilla kanavan tuottaman bittivirhesuhteen arvosta $f = 0{,}1$ arvoon $\text{BER} \approx 0{,}064$. Tästä maksamme sen, että kanavaa on käytettävä 7/4 -kertaisesti saman informaation välittämiseen, eli kanavassa liikkuu jokaista lähetettyä seitsemää bittiä kohden vain neljä bittiä informaatiota. Sanomme, että Hamming(7,4)-koodin *informaatiosuhde* (engl. rate) on $R = 4/7$.

Hamming-koodeja on olemassa useita muitakin, ja ne muodostavat itse asiassa kokonaisen *koodiperheen*, jossa jokaisen koodin viestien pituus on $2^r - r - 1$ ja koodisanojen pituus $2^r - 1$, missä $r \in \{2, 3, 4, \ldots\}$. Kaikki Hamming-koodit pystyvät korjaamaan varmuudella tasan yhden virheen koodisanaa kohti. Hamming(7,4)-koodin laajentamista tähän muotoon ei käsitellä tässä työssä tarkemmin, mutta siihen voi perehtyä esimerkiksi kirjan [11, s. 244 – 282] avulla.

Hamming-koodien virheenkorjauskyky perustuu siis koodisanojen eroavaisuuksiin. Koodisano-

ja hyödyntävissä koodeissa eli *lohkokoodeissa* koodisanojen pienintä eroa toisistaan, mitattuna eroavien bittien määrällä, kutsutaan koodin *etäisyydeksi* d. Esimerkiksi Hamming-koodien etäisyys $d = 3$ ja r-kertaisen toistokoodin, jossa koodisanat saadaan yksinkertaisesti toistamalla jokainen bitti r kertaa, etäisyys on $d = r$. Etäisyys on sikäli tärkeä ominaisuus, että sen avulla nähdään suoraan koodin pystyvän korjaamaan varmuudella $\frac{d-1}{2}$ virhettä jokaisesta lohkosta. Etäisyys siis liittyy bittivirhesuhteeseen. Se on myös käytännöllinen tapa kuvata koodia, sillä se ei riipu kanavasta ja sen laskeminen on suhteellisen helppoa. Niinpä määrittelemme seuraavaksi ensimmäisen mittarimme koodin hyvyydelle.

Määritelmä 2.8. Koodiperheen koodien etäisyys on

- *hyvä*, jos suhteellinen etäisyys d/n lähestyy nollaa suurempaa vakiota,
- *huono*, jos d/n lähestyy nollaa mutta etäisyys d ei lähesty vakiota,
- *erittäin huono*, jos etäisyys d lähestyy vakiota

koodisanojen pituuden n kasvaessa. [15]

Esimerkiksi toistokoodin etäisyys on hyvä, sillä sen kaksi koodisanaa eroavat toisistaan jokaisessa bitissään. Toisaalta hyvän bittivirhesuhteen lisäksi haluaisimme myös suuren informaatiosuhteen, jotta viestin välittämiseen ei kuluisi liikaa aikaa ja energiaa. Tämä huomioiden toistokoodit eivät vaikuta enää niin hyviltä, sillä siinä missä niiden etäisyys $d = r$ nousee voimakkaasti, niiden informaatiosuhde $R = k/n = 1/r$ laskee samassa suhteessa. Itse asiassa voidaan osoittaa [23], että kaikille lohkokoodeille pätee

$$\frac{k}{n} + \frac{d}{n} \leq 1 + \frac{1}{n}. \tag{2.4}$$

Tätä kutsutaan Singleton-rajaksi, ja sen yhtäsuuruuden toteuttavia koodeja MDS (engl. maximum distance separable) -koodeiksi, joista tavallisin epätriviaali esimerkki ovat Reed-Solomon -koodit [19]. Vastaavanlaisia lohkokoodien ominaisuuksia koskevia epäyhtälöitä on muitakin, mutta näitä ei käsitellä tässä työssä enempää.

Olemme nyt tutustuneet lohkokoodien peruskäsitteisiin ja todenneet, että niiden helposti laskettavaa etäisyysominaisuutta voidaan käyttää indikaattorina niiden virheenkorjauskyvylle. Tähän perustuen loimme konkreettisen mittarin määritelmässä 2.8. Totesimme myös koodin suorituskyvyn kokonaisuudessaan riippuvan etäisyyden lisäksi informaatiosuhteesta, ja näiden kahden

olevan tiiviisti sidottu toisiinsa Singleton-rajan kautta. Tämä saa ehkä hyvien koodien kehittämisen vaikuttamaan nollasummapeliltä, sekä koodin suorituskyvyn arvioinnin vaikuttamaan vain sen selvittämiseltä, missä suhteessa koodissa painottuu hyvä etäisyys informaatiosuhteen kustannuksella. Todellisuudessa tilanne ei ole lainkaan näin huono, sillä pystymme saavuttamaan mitättömän pieniä bittivirhesuhteita hyvillä informaatiosuhteilla ja jopa koodeilla, joiden etäisyydet ovat määritelmämme mukaan huonoja. Tämä johtuu siitä, että etäisyys on nopean päättelymme vastaisesti varsin huono koodin suorituskyvyn mittari, vaikka siihen varsin usein keskitytäänkin [15, s. 206]. Seuraavassa alaluvussa paneudumme tähän ristiriitaan ja määrittelemme paremman mittarin koodin suorituskyvylle.

2.3 Kanavan kapasiteetti

Edellisessä alaluvussa tarkastelimme FEC-algoritmien suorituskykyjä, päätyen määrittelemään perinteisen ja luonnolliselta tuntuvan mittarin näiden virheenkorjauksen tehokkuudelle käyttäen indikaattorina niiden etäisyysominaisuutta. Ymmärtääksemme lähestymistavan heikkouden ja johtaaksemme paremman mittarin, tarvitsemme selkeän kuvan siitä, millainen suorituskyky on ylipäätään saavutettavissa kommunikaatiossa häiriöisen kanavan läpi. Avainasemassa on Claude Shannonin esittämä ja todistama häiriöisen kanavan koodauslause, jonka ymmärtämiseksi tarvitsemme vielä informaatioteorian peruskäsitteistä entropian ja yhteisinformaation.

Määritelmä 2.9. Satunnaismuuttujien X ja Y välinen *yhteisinformaatio* on

$$I(X;Y) = H(X) - H(X\,|\,Y) = H(Y) - H(Y\,|\,X). \tag{2.5}$$

Määritelmässä esiintyy entropia $H(\cdot) = \int_{\mathscr{A}} p(\cdot)\log\frac{1}{p(\cdot)}\,\mathrm{d}\cdot$, missä integraali on jakauman kantajan yli, logaritmi kaksikantainen ja yksikkö bitti. Tämä mittaa epävarmuutta. Esimerkiksi tavallisen kolikonheiton entropia on $\frac{1}{2}\cdot\log 2 + \frac{1}{2}\cdot\log 2 = 1$ bitti, mutta jos tiedämme kolikkoa taivutetun ja sen tuottavan kruunan vain joka neljännellä heitolla, on sen entropia vain $\frac{1}{4}\log 4 + \frac{3}{4}\log\frac{4}{3} \approx 0{,}81$ bittiä. Mitä parempi on mahdollisuutemme veikata tulos oikein, sitä vähemmän siihen liittyy epävarmuutta.

Kun tulkitsemme kanavaan lähetettävän viestin sekä vastaanotetun viestin satunnaismuuttujiksi, yhteisinformaatio kertoo siis määritelmänsä mukaan, miten paljon alkuperäistä viestiä koskeva epävarmuus vastaanottopäässä keskimäärin vähenee, kun tämä vastaanotettu viesti havaitaan. Lienee luonnollinen ajatus, että kanava on sitä parempi, mitä enemmän tämä epävarmuus vä-

henee, eli mitä suurempi lähetetyn ja vastaanotetun viestin yhteinen informaatio on. Sivuhuomiona yhteisinformaatio voi riippua kanavan lisäksi vain lähetyksissä käytettävien symbolien jakaumasta $\mathscr{P}_X$. Voidaan todeta melko helposti, että esimerkiksi binäärisen symmetrisen kanavan tapauksessa se on suurin, kun kumpaakin symbolia käytetään koodissa keskimäärin yhtä paljon.

Seuraavaksi voimme esittää tärkeän tuloksen, joka muuttaa edellisessä alaluvussa esitetyn suhtautumisen koodien suorituskykyjen saavutettavuuteen. Lauseen todistus on esitetty esimerkiksi kirjassa [15, s. 81–83].

Lause 2.10. *Shannonin häiriöisen kanavan koodauslause*.

1. Diskreetin, muistittoman kanavan *kapasiteetilla*

$$C = \max_{\mathscr{P}_X} I(X;Y) \tag{2.6}$$

   on seuraava ominaisuus: kaikille $\varepsilon > 0$ ja $R < C$ on olemassa riittävän suuri $n \in \mathbb{N}$ siten, että on olemassa koodi, jonka koodisanojen pituus on n ja informaatiosuhde $k/n \geq R$ siten, että lohkovirheen mahdollisuus on pienempi kuin ε.

2. Jos bittivirheen todennäköisyys p_b on hyväksyttävä, informaatiosuhteet aina arvoon $R(p_b)$ saakka ovat saavutettavissa, missä

$$R(p_b) = \frac{C}{1 - \left(p_b \log\left(\frac{1}{p_b}\right) + (1-p_b)\log\left(\frac{1}{1-p_b}\right)\right)}. \tag{2.7}$$

3. Mille tahansa bittivirheen todennäköisyydelle p_b, arvoa $R(p_b)$ suuremmat informaatiosuhteet eivät ole saavutettavissa.

Esimerkki 2.11. Kuvassa 1 on esitetty r-kertaisten toistokoodien suorituskykyjä binäärisessä symmetrisessä kanavassa virhetiheydellä $f = 0{,}1$. Tämän kanavan kapasiteetti on

$$C = \max_{\mathscr{P}_X} I(X;Y) = \max_{\mathscr{P}_X} \left(H(Y) - H(Y\,|\,X)\right) \approx 0{,}53,$$

joten lauseen 2.10 mukaan on olemassa lohkokoodeja, joiden informaatiosuhde on $\frac{1}{2}$ ja bittivirhesuhde niin pieni, kuin vain haluamme. Esimerkiksi bittivirhesuhteen $\mathrm{BER} = 1 \cdot 10^{-5}$ saavuttamiseen toistokoodilla tarvitsisimme vähintään 19-kertaisen toistokoodin, jolloin informaatiosuhde olisi vain noin 0,053. Pienempiäkin bittivirhesuhteita saavutetaan siis lauseen 2.10 mukaan lähes kymmenkertaisella informaatiosuhteella.

Palataksemme tämän luvun alkuperäiseen ongelmaan, haluamme löytää keinon FEC-algoritmien suorituskyvyn tutkimiseen. Suorituskyvyllä tarkoitamme mahdollisimman pientä bittivirhesuhdetta mahdollisimman suurella informaatiosuhteella, ja tämän tiedämme nyt olevan rajoitetun lauseen 2.10 mukaisesti. Toisaalta sama lause myös lupaa, että hyvin lähelle tätä rajaa pääseviä lohkokoodeja on olemassa, mutta se ei kerro miten näitä voitaisiin löytää.

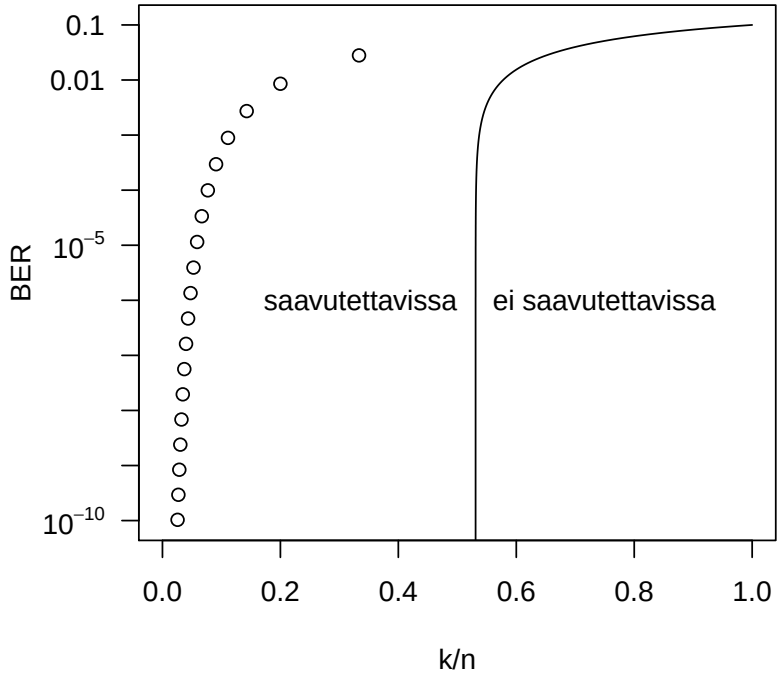
Lohkokoodin etäisyyteen d keskittyminen ei ole hyvien koodien kehittämisen kannalta hedelmällistä, sillä etäisyys määrittää ylärajan vain *täysin varmasti* korjattavien virheiden lukumäärälle. Suurten lohkokokojen koodeissa pystymme korjaamaan lukuisia virheitä hyvin todennäköisesti, joskaan emme täysin varmasti. Itse asiassa tutkimalla n-pituisia koodisanoja mahdollisine virheineen kuvan 2 tapaan n-ulotteisena avaruutena, jossa varmasti dekoodaantuvat sanat muodostavat kuulat sallittujen koodisanojen ympärille, voidaan todeta varmasti dekoodaantuvien sanojen suhteen muuhun avaruuteen lähestyvän nollaa pituuden n kasvaessa [22]. Hyvän FEC-algoritmin on siis korjattava paljon muutakin kuin “varmat” tapaukset.

Muotoilemme seuraavaksi uuden mittarin FEC-algoritmin suorituskyvylle perustaen sen lauseeseen 2.10.

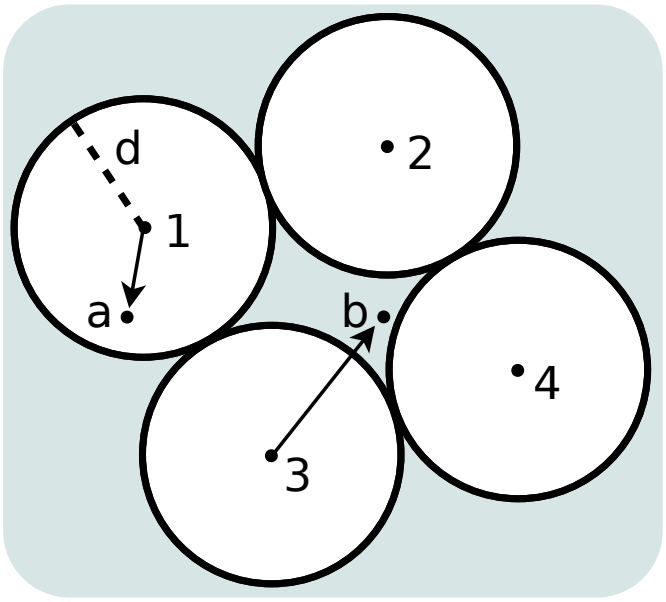
Määritelmä 2.12. Koodiperhe on jonkin kanavan suhteen

- *erittäin hyvä*, jos sillä voidaan saavuttaa mielivaltaisen pieni bittivirhesuhde millä tahansa informaatiosuhteella aina kanavan kapasiteettiin saakka,
- *hyvä*, jos sillä voidaan saavuttaa mielivaltaisen pieni bittivirhesuhde millä tahansa informaatiosuhteella aina johonkin maksimiin saakka, joka voi olla pienempi kuin kanavan kapasiteetti,
- *huono*, jos sillä ei voida saavuttaa mielivaltaisen pientä bittivirhesuhdetta tai se saavutetaan vain informaatiosuhteen lähestyessä nollaa. [15]

Tämän määritelmän etuna on sen nojaaminen Shannonin todistamaan suorituskykyrajaan. Toisaalta se mittaa nimenomaan koodiperheen suorituskykyä koodisanojen pituutta kasvatettaessa, joten periaatteessa hyväksi todetun koodiperheen käytännössä toteuttamiskelpoiset koodit voivat olla suorituskyvyltään huonoja, jos niitä ei pystytä toteuttamaan tarpeeksi pitkillä koodisanoilla. Hyvän FEC-algoritmin etsintä kannattaa siis aloittaa hyväksi tai erittäin hyväksi osoitetusta perheestä, valiten mahdollisimman suuri koodisanojen pituus, mutta yksittäisen algoritmin suorituskyky on vielä testattava erikseen varmuuden saamiseksi.



Kuva 1: Lauseen 2.10 määrittämä suorituskykyraja binääriselle symmetriselle kanavalle häiriösuhteella $f = 0{,}1$. Parittomien toistomäärien toistokoodien virhesuhteet on merkitty kuvaan ympyröillä, alkaen 3-kertaisesta toistokoodista, joka on kuvassa ylimpänä.



Kuva 2: Neljä koodisanaa kaksiulotteisessa avaruudessa. Tiedonsiirron aiheuttamat virheet poikkeuttavat lähetettyä koodisanaa, mutta jos se siirtyy enintään etäisyyden d verran, se voidaan vielä dekoodata oikein varmuudella. Koodin etäisyyden määrittämä dekoodaantuva alue on merkitty kuvassa valkoisella. Esimerkiksi vastaanotettu koodisana a dekoodaantuisi vielä, mutta b ei. Ongelmallista on, että siirryttäessä korkeampiin ulottuvuuksiin, harmaan alueen suuruus suhteessa valkoiseen kasvaa rajatta. Tämän vuoksi dekoodausta ei välttämättä kannata perustaa pelkkään koodin etäisyyteen.

2.4 Koodausvahvistus

Tässä alaluvussa esitellään edellisiin verrattuna varsin konkreettinen mittari yksittäisen koodin suorituskyvylle. Se perustuu vertailujen tekemiseen johonkin toiseen koodiin tai koodaamattomaan yhteyteen. Huonona puolena se on periaatteeltaan edellisiä monimutkaisempi, sillä joudumme huomioimaan kanavan modulaatiotekniikan. Lisäksi siinä esiintyvä suhde E_b/N_0 voi jäädä abstraktiksi käsitteeksi, ellei ymmärrä taustalla olevaa matematiikkaa. Tätä mittaria kuitenkin käytetään varsin yleisesti, sillä se on koodin etäisyyttä tarkempi ja koodiperheen hyvyyttä spesifimpi.

Kanavan, esimerkiksi radioyhteyden, bittivirhesuhdetta voidaan usein pienentää käyttämällä enemmän lähetystehoa tai suurempaa kaistanleveyttä. Jos valitaan bittivirhesuhteelle jokin hyväksyttävä taso, se saavutetaan virheenkorjauskoodia käyttäen käytännössä pienemmällä lähetysteholla. Virheenkorjauskoodin voidaan siten katsoa antavan lähetystehon nostamiseen verrattavissa olevan hyödyn, *koodausvahvistuksen*. Koodausvahvistus voidaan käyttää esimerkiksi pidemmän yhteysvälin toteuttamiseen tai marginaalina häirintää vastaan [13].

Radioyhteyden bittivirhesuhde riippuu signaali-kohinasuhteen lisäksi käytetystä modulaatiomenetelmästä. Koodausvahvistus onkin nähtävissä nimenomaan modulaatiomenetelmää vastaavasta taulukosta tai kaaviosta, kun valitaan jokin bittivirhesuhteen taso. Seuraavissa esimerkeissä lasketaan bittivirhesuhteet kahdelle modulaatiomenetelmälle sekä havainnollistetaan, miten koodausvahvistus on tulkittavissa niitä vastaavista kaavioista.

Seuraavat bittivirhesuhdelaskut perustuvat jatkuva-aikaisen kanavan esittämiseen diskreettiaikaisena AWGN-kanavana. Jatkuva-aikaisessa kanavassa voidaan lähettää N reaalilukua $\{x_n\}_{n=1}^N$ T-kestoisena signaalina, joka muodostetaan ortonormaaleiden kantafunktioiden $\phi_n(t)$ painotettuna kombinaationa

$$x(t) = \sum_{n=1}^{N} x_n \phi_n(t). \tag{2.8}$$

Signaalin kuljettua kanavan läpi, se voidaan purkaa:

$$y_n = \int_0^T \phi_n(t) y(t) \mathrm{d}t = x_n + \int_0^T \phi_n(t) n(t) \mathrm{d}t = x_n + n_n \tag{2.9}$$

kaikilla $n = 1, \ldots, N$. Kohina $n(t)$ siis lisää estimaattiin y_n skalaarisen, normaalijakautuneen kohinan $n_n \sim N(0, N_0/2)$, missä N_0 on jatkuva-aikaisen kanavan kohinan spektraalitiheys. Tämä on käsitelty yksityiskohtaisemmin esimerkiksi kirjassa [15, s. 177–178].

Esimerkki 2.13. *Bittivirhesuhde BPSK-modulaatiolle.* BPSK (Binary Phase Shift Keying) -modulaatio perustuu viestittämiseen yhtä kantafunktiota käyttäen, muuttaen signaalin vaihetta lähetettävien bittien mukaisesti. Signaali noudattaa kaavaa

$$s_n(t) = \sqrt{\frac{2E_b}{T_b}}\cos(2\pi f_c t + \pi(1-n)), \;\; n \in \{0,1\}, \tag{2.10}$$

missä E_b on energia bittiä kohden, T_b yhden bitin kesto ja f_c kantoaallon taajuus. Signaalilla on kantafunktio

$$\phi(t) = \sqrt{\frac{2}{T_b}}\cos(2\pi f_c t), \tag{2.11}$$

bittien ollessa $\sqrt{E_b}\phi(t)$ ja $-\sqrt{E_b}\phi(t)$. BPSK-modulaation periaatetta on havainnollistettu kuvassa 3.

BPSK-modulaatiota vastaa AWGN-kanava biteillä $\sqrt{E_b}$ ja $-\sqrt{E_b}$. Kanava lisää jokaiseen lähetettyyn bittiin kohinan $n_n \sim N(0, N_0/2)$. Tämän symmetriasta johtuen dekoodaus voidaan suorittaa yksinkertaisesti tarkistamalla vastaanotetun skalaarin $y_n = x_n + n_n$ etumerkki. Virhe syntyy, jos kohina vähentää positiivista signaalia enemmän kuin $\sqrt{E_b}$:n verran tai lisää negatiivista signaalia vastaavasti. Tämän todennäköisyys on helposti laskettavissa, ja tulokseksi saadaan

$$p_b = \frac{1}{2}\operatorname{erfc}\left(\sqrt{\frac{E_b}{N_0}}\right). \tag{2.12}$$

Esimerkki 2.14. *Bittivirhesuhde BFSK-modulaatiolle.* BFSK (Binary Frequency Phase Shift Keying) -modulaatio perustuu viestittämiseen kahta eritaajuista kantoaaltoa käyttäen, vaihtaen lähetettävää taajuutta bittien mukaisesti. Signaali noudattaa kaavaa

$$s_n(t) = \sqrt{\frac{2E_b}{T_b}}\cos(2\pi f_n t), \;\; n \in \{0,1\}. \tag{2.13}$$

Toisin ilmaistuna signaalilla on siis kantafunktiot

$$\phi_1(t) = \sqrt{\frac{2}{T_b}}\cos(2\pi f_1 t) \;\text{ ja} \tag{2.14}$$

$$\phi_2(t) = \sqrt{\frac{2}{T_b}}\cos(2\pi f_2 t), \tag{2.15}$$

bittien ollessa $\sqrt{E_b}\phi_1(t)$ ja $\sqrt{E_b}\phi_2(t)$. Kantafunktioiden vaihe-ero tunnetaan, joten signaali voidaan demoduloida koherentisti. Lisäksi vaatimuksestamme kantafunktioiden ortonormaaliudel-

le seuraa, että taajuuksien f_1 ja f_2 on oltava jaollisia luvulla $4T_b$, ja taajuuksien välisen etäisyyden Δf luvulla $2T_b$ [11]. BFSK-modulaation periaatetta on havainnollistettu kuvassa 4.

BFSK-modulaation esittäminen AWGN-kanavan avulla vaatii hieman enemmän edellisen esimerkin tilanteeseen verrattuna. Nyt kantafunktioita on kaksi, joten signaalien välistä etäisyyttä ei voi laskea yksinkertaisesti vähennyslaskulla. On otettava huomioon kantafunktioiden ortogonaalisuus, jolloin signaalien väliseksi etäisyydeksi saadaan, kuten kuvasta 5 on nähtävissä, $\sqrt{2E_b}$. Signaalit ovat yhtä voimakkaat, joten vastaavan AWGN-kanavan biteiksi saadaan $\pm\frac{1}{2}\sqrt{2E_b} = \pm\sqrt{\frac{E_b}{2}}$. Muutoin täysin edellistä esimerkkiä vastaavalla päättelyllä saadaan BFSK-modulaation bittivirhesuhteeksi

$$p_b = \frac{1}{2}\operatorname{erfc}\left(\sqrt{\frac{E_b}{2N_0}}\right). \tag{2.16}$$

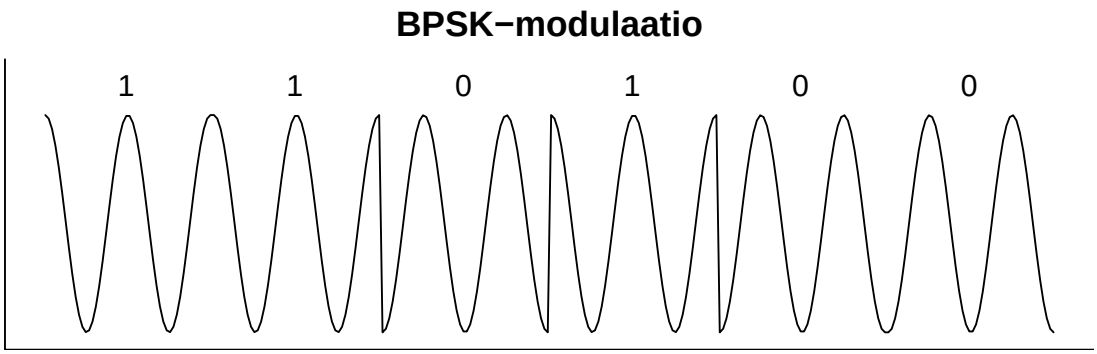
Bittivirhesuhde piirretään tavallisesti edellisissäkin esimerkeissä esiintyneen suhteen E_b/N_0 funktiona. Moduloitu mutta koodaamaton signaali on hyvä vertailukohta koodatulle signaalille. Kuvaan 6 on piirretty BPSK- sekä BFSK-modulaatioiden bittivirhesuhteet koodaamattomina sekä Hamming(7,4) -koodauksella. Koodausvahvistus luetaan valitsemalla kiinteä bittivirhesuhde ja tutkimalla kuvaajien erotusta. Esimerkiksi bittivirhesuhteella 10^{-9} Hamming-koodi tuottaa noin 3 dB vahvistuksen. Kuvasta nähdään, että myös BPSK tarjoaa BFSK:n nähden noin 3 dB vahvistuksen.

Koodausvahvistus on mittarina sikäli yksipuoleinen, ettei se huomioi koodin informaatiosuhdetta. Kuten aiemmin totesimme, voidaan esimerkiksi toistokoodilla saavuttaa hyvä virheenkorjauskyky, mutta erittäin huonolla informaatiosuhteella. Kuvaan 6 onkin piirretty myös 21-kertainen toistokoodi, joka voisi pelkästään tätä kuvaa katsomalla vaikuttaa huomattavasti Hamming(7,4) -koodia paremmalta ratkaisulta. Niiden informaatiosuhteet ovat kuitenkin aivan eri luokkaa. Kuvan 6 kaltaisia käyriä tulkittaessa onkin varmistuttava siitä, että verrattavien koodien informaatiosuhteet ovat keskenään vertailukelpoisia.

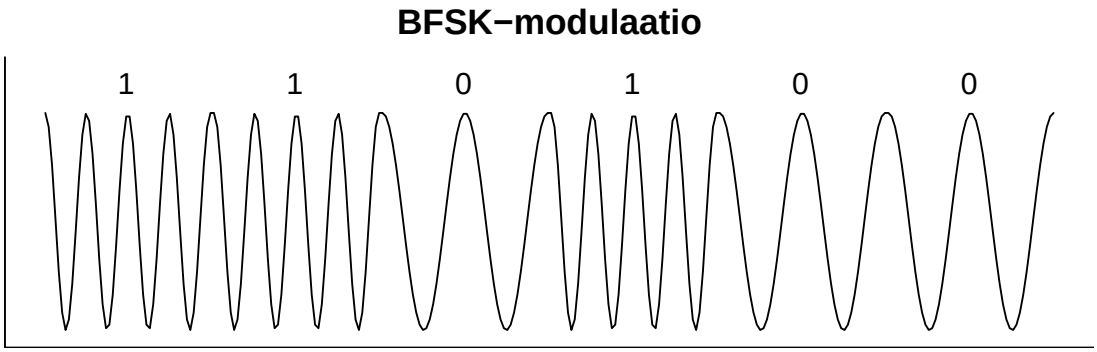
Mielekkäämpään vertailuun myös informaatiosuhteiltaan erilaisten koodien välillä päästään, kun vakioimme tarkastelussa bittivirhesuhteen jollekin (mielellään realistiselle) tasolle, jolloin voimme ottaa informaatiosuhteen toiseksi muuttujaksi. Huomioimme vielä modulaation vaikutuksen tarkastelemalla sitä, miten monta bittiä informaatiota järjestelmämme välittää yhtä kanavan käyttöä kohti. Tätä kutsutaan spektraaliseksi tehokkuudeksi ja sen yksikkö on (bittiä/s)/Hz. Esimerkiksi BPSK-modulaatiossa on kaksi mahdollista tilaa, joten sen spektraalinen tehokkuus on $\log_2 2 = 1$, kerrottuna käytetyn kanavakoodauksen informaatiosuhteella.

Kuvasta 7 löytyvät samat Hamming(7,4) ja 21-kertainen toistokoodi, joita tarkastelimme myös kuvassa 6, piirrettynä tällä kertaa informaatiosuhde huomioituna edellä kuvatulla tavalla. Kuvasta 7 nähdään, ettei 21-kertainen toistokoodi ole ainakaan selvästi parempi. Itse asiassa Hamming(7,4) pääsee huomattavasti lähemmäs lauseen 2.10 määrittämää rajaa, eli kanavan kapasiteettia, vaikka häviääkin virheenkorjauskyvyltään. Toisaalta 21-kertaisen toistokoodin informaatiosuhde on niin huono, ettei sitä useimmissa todellisissa sovelluksissa kannattaisi edes harkita. Toistokoodeista lähimpänä Hamming(7,4) -koodia on lyhyin, eli 3-kertainen toistokoodi, joka taas häviää sille selvästi.

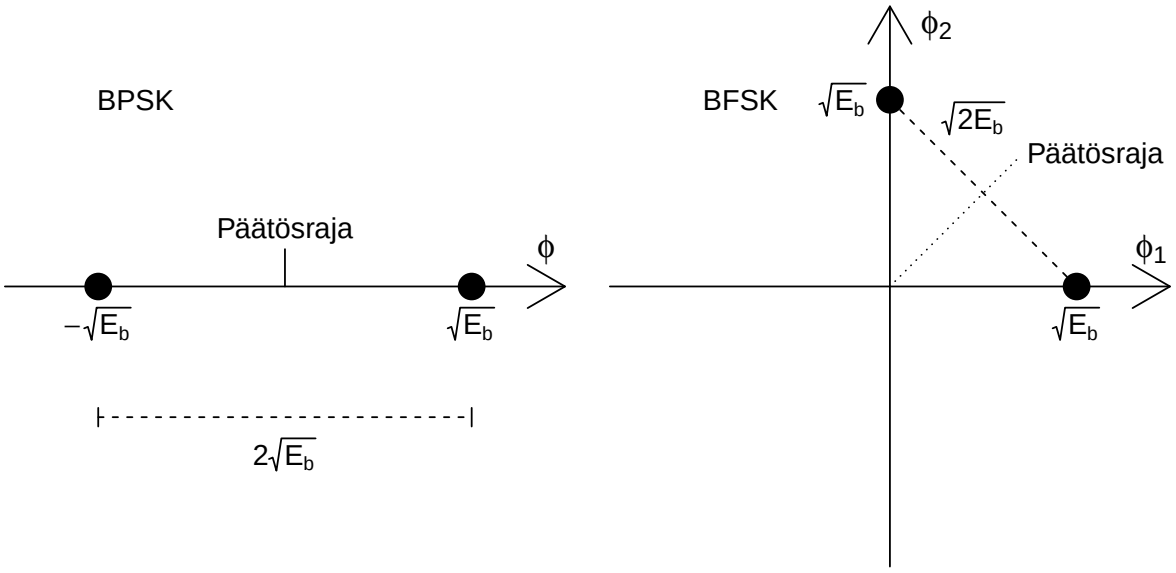
Kuvassa 7 yksinkertaisten Hamming- ja toistokoodien sekä kanavan kapasiteetin määrittämän rajan väliin jää paljon tilaa tehokkaammille virheenkorjauskoodeille. Lause 2.10 kuitenkin vain asettaa teoreettisen rajan koodien suorituskyvylle ja todistaa tällaisia koodeja olevan olemassa, antamatta sellaisesta yhtään esimerkkiä. Parempia algoritmeja on kehitetty Hamming-koodien keksimisestä lähtien, ja tämä kehitystyö onkin tuonut suorituskyvyn jo varsin lähelle lauseen 2.10 määrittämää rajaa. Seuraavassa luvussa esittelemme tärkeimmät virheenkorjauskoodien luokat ja nykyaikaisimpien koodien toimintaperiaatteet. Luvun lopussa on kuvaa 7 vastaava kuva, johon on lisätty näistä esimerkit.



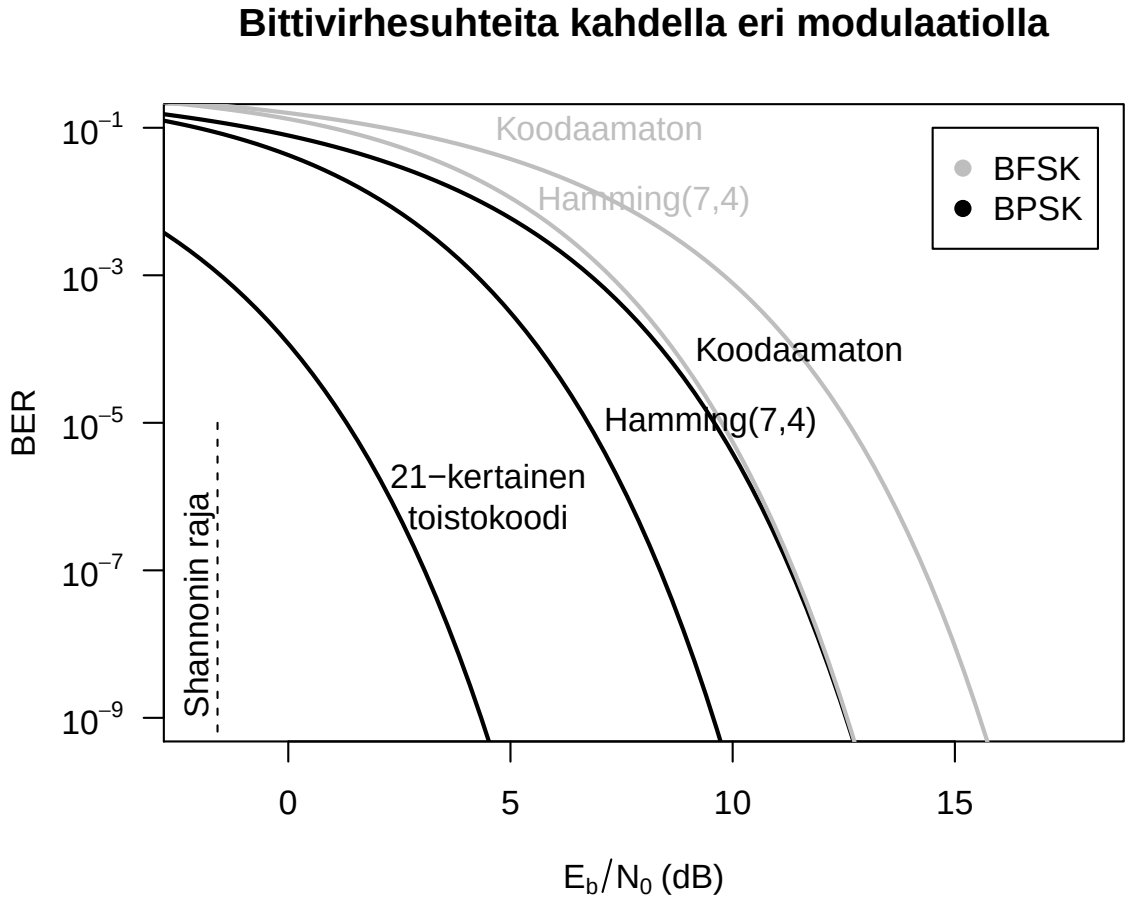
Kuva 3: BPSK-modulaatiossa bitit viestitetään signaalin vaiheen avulla. Vaihtoehtoisesti voitaisiin koodata esimerkiksi jokainen bitti 1 vaiheen muutoksella, jolloin signaali voitaisiin demoduloida myös epäkoherentisti. Kyseessä olisi tällöin DPSK (Differential Phase-Shift Keying).



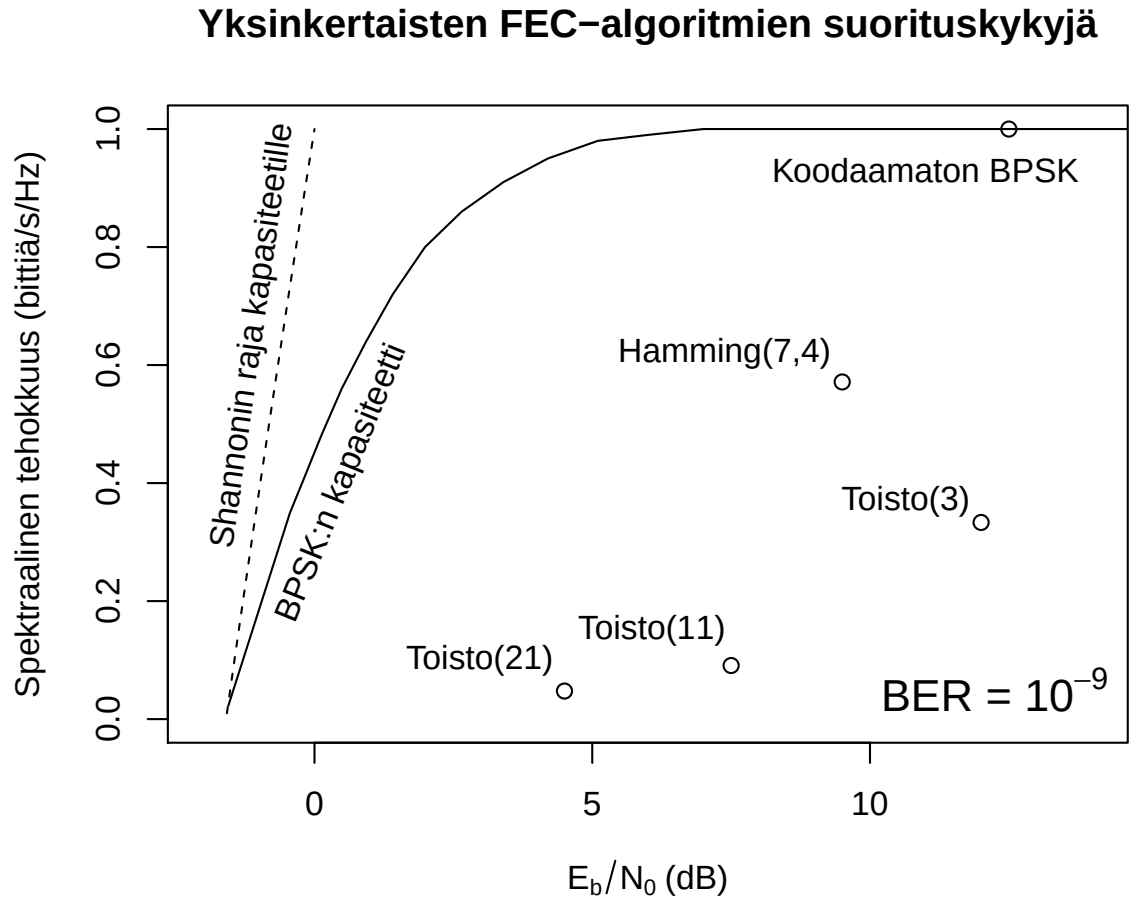
Kuva 4: BFSK-modulaatiossa bitit viestitetään eritaajuisina signaaleina. Kuvan esimerkkisignaalin vaihe ei hyppää missään vaiheessa (CPFSK, Continuous-Phase Frequency-Shift Keying), mikä voidaan käytännössä toteuttaa käyttämällä yhtä oskillaattoria, jonka taajuutta voidaan muuttaa. Tämä takaa mahdollisuuden käyttää joko koherenttia tai epäkoherenttia demodulaatiota.



Kuva 5: BPSK käyttää saman kantafunktion vastakkaisia vaiheita, BFSK kahta kantafunktiota. Lähtenyt bitti dekoodaantuu oikein, kunhan se ei harhaudu päätösrajan toiselle puolelle. BPSK-modulaatiossa etäisyys päätösrajaan on $\sqrt{E_b}$, BFSK-modulaatiossa $\frac{1}{2}\sqrt{2E_b} = \sqrt{E_b/2}$.



Kuva 6: Bittivirhesuhteita kohinan spektraalitiheydellä jaetun bittikohtaisen energian funktiona. Koodausvahvistus luetaan valitsemalla tietty bittivirhesuhde ja vertaamalla käyrien erotusta tällä suoralla. Esimerkiksi tasolla BER $= 10^{-9}$ Hamming(7,4) -koodin vahvistus koodaamattomaan kanavaan verrattuna on n. 3 dB (molemmilla modulaatioilla) ja 21-kertaisen toistokoodin n. 8 dB. Jälkimmäisen informaatiosuhde on kuitenkin huomattavasti Hamming-koodia huonompi, mitä koodausvahvistus ei huomioi. Kuvassa on näkyvissä myös Shannonin-Hartleyn lain määrittämä raja (n. -1.59 dB), jota pienemmällä E_b/N_0 -suhteella mielivaltaisen pieni bittivirhesuhde ei ole missään tapauksessa saavutettavissa [25].



Kuva 7: Kanavakoodauksien spektraalisia tehokkuuksia kohinan spektraalitiheydellä jaetun bittikohtaisen energian funktiona, kun bittivirhesuhde on kiinnitetty arvoon 10^{-9} ja modulaatiomenetelmä on BPSK. Koodi on sitä parempi, mitä pienemmällä suhteella E_b/N_0 se voi toimia, säilyttäen silti suuren informaatiosuhteen ja sitä kautta spektraalisen tehokkuuden. Tälle antaa rajan lauseen 2.10 mukaisesti kanavan kapasiteetti, jota taas rajaa Shannonin-Hartleyn laki [11]. Toistokoodit sekä Hamming(7,4) jäävät kauas kapasiteetista, erityisesti toistokoodin kehnous näkyy huomattavasti kuvaa 6 paremmin.

3 TÄRKEIMMÄT VIRHEENKORJAUSALGORITMIT

Tässä luvussa esitellään tärkeimmät FEC-algoritmien tyypit ominaispiirteineen. Aloitamme tutustumalla perinteiseen kahtiajakoon *lineaarisiin lohkokoodeihin* sekä *konvoluutiokoodeihin*. Edellisiin kuuluu muun muassa ensimmäisenä varsinaisena FEC-algoritmina pidettävä, Richard Hammingin vuonna 1950 julkaisema Hamming(7,4) -algoritmi [12]. Jälkimmäisten perusajatus esitettiin vain viisi vuotta myöhemmin, ja nämä yhdessä määrittävät edelleen perustavanlaatuisen jaottelun.

FEC-algoritmien jaottelun perusteiden jälkeen luvussa tarkastellaan useamman koodin yhdistämisellä *ketjukoodiksi* saavutettavia etuja. Nämä kehitettiin 1960-luvulla, ja ne olivat tehokkaimpia tunnettuja ratkaisuja häiriöttömän kommunikaation ongelmaan aina 1990-luvulle saakka, jolloin löydettiin suorituskyvyltään ylivertaiset *turbokoodit* (1993) sekä *harvat parillisuustarkistuskoodit* (engl. Low-Density Parity-Check, LDPC) (1994–1995[1]) [7]. Nämä ovat johdannaisineen parhaita löydettyjä FEC-algoritmeja, kuten tulemme havaitsemaan luvun lopussa niitä käsiteltäessä. [10, 14]

Jokaisen koodityypin yhteydessä tarkastellaan myös dekoodausta, sillä se on usein monimutkaisuutensa vuoksi itse koodausta merkittävämpi rajoite käytettävyydelle.

3.1 Lineaariset lohkokoodit

Luvussa 2 johdimme koodausteorian peruskäsitteitä lohkokoodiesimerkkien avulla. Määrittelimme lohkokoodin virheenkorjausalgoritmiksi, jossa lähetetyt bitit jaetaan k-pituisiksi viesteiksi, ja jotka lähetetään kanavaan n-pituisina koodisanoina. Koodin etäisyyden d määrittelimme näiden koodisanojen pienimmäksi etäisyydeksi toisistaan, mitattuna eroavien bittien määrässä. Nämä luvut kuvaavat lohkokoodeja sikäli merkittävästi, että usein puhutaankin (n,k)- tai (n,k,d)-lohkokoodeista ja konkreettiset algoritmit kirjoitetaan esimerkiksi Hamming(7,4) tai Hamming(7,4,3), missä luvut ovat vastaavien suureiden arvot.

Lineaariset lohkokoodit ovat lohkokoodien merkittävä erikoistapaus, joilla kaikkien koodisanojen lineaarikombinaatiot ovat myös koodisanoja. Koodisanojen joukolla on tällöin hyviä matemaattisia ominaisuuksia, joista on apua sekä koodauksessa että dekoodauksessa. Lineaarisen (n,k)-koodin koodisanat muodostavat nimittäin vektoriavaruuden $\{0,1\}^n$ k-dimensioisen

[1] Tosiasiassa LDPC-algoritmin julkaisi Robert Gallager jo vuonna 1963, mutta sitä ei kyetty toteuttamaan käytännössä aikansa laitteistoilla. Se jäi unohduksiin ja löydettiin uudestaan 1990-luvulla.

lineaarisen aliavaruuden C, jolloin koodisana $\mathbf{s}$ voidaan muodostaa viestistä $\mathbf{m}$ matriisitulolla $\mathbf{s} = \mathbf{mG}$, jossa $\mathbf{G}$ on koodin *generaattorimatriisi*.

Esimerkiksi tuttu esimerkkikoodimme Hamming(7,4) on lineaarinen, ja se voidaan määritellä luvun 2 määritelmän sijaan generaattorimatriisilla

$$\mathbf{G} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix},$$

jolloin jokainen koodisana saadaan taulukosta katsomisen sijaan laskemalla yksi matriisitulo. Esimerkiksi kun viesti $m = [0001]^T$, niin koodisana $\mathbf{s} = \mathbf{mG} = [0001011]^T$, mikä on sama kuin määritelmän 2.6 mukaan.

Hamming(7,4) koodissa on vain $2^4 = 16$ koodisanaa, jolloin tällä mahdollisuudella ei ole vielä merkitystä, mutta jos lineaarisen lohkokoodin viestien pituus olisi esimerkiksi $k = 300$ bittiä, tarvittaisiin $2^{300} \approx 2 \cdot 10^{90}$-rivinen taulukko. Vertailun vuoksi koko havaittavassa maailmankaikkeudessa on vain noin 10^{80} atomia, joten tämän koodin määritteleminen taulukon sijaan 300-rivisellä matriisilla olisi huomattavan käytännöllistä. Lineaaristen lohkokoodien merkittävyys perustuu koodauksen osalta siihen, että koodin määritteleminen matriisilla on käytännöllistä jo kohtuullisillakin viestien pituuksilla k. Toisaalta lauseen 2.10 perusteella hyvissä koodeissa viestien pituus k on nimenomaan suuri.

Lineaaristen lohkokoodien toinen merkittävä ominaisuus liittyy dekoodaukseen. Määritelmässä 2.6 dekoodaus tapahtuu etsimällä koodisana, joka eroaa vastaanotetusta lohkosta korkeintaan yhdessä bitissä. Normaalisti tämä tarkoittaisi kaikkien taulukon rivien läpikäymistä ja vertaamista vastaanotettuun lohkoon kunnes hyväksyttävä koodisana löytyy, mikä on jälleen hidasta suurilla taulukoilla. Lineaariset lohkokoodit voidaan kuitenkin dekoodata *syndroomadekoodauksella*, joka on lineaarisen koodin matemaattisia ominaisuuksia hyödyntävä, huomattavasti tehokkaampi menetelmä.

Kun generaattorimatriisi esitetään muodossa $\mathbf{G} = [\mathbf{I}_k|\mathbf{P}]$, missä $\mathbf{I_k}$ on $k \times k$-dimensioinen yksikkömatriisi, niin matriisia $\mathbf{H} = [\mathbf{P}^T|\mathbf{I}_{n-k}]$ kutsutaan koodin *parillisuustarkistusmatriisiksi*. Sen ominaisuutena kaikilla koodin koodisanoilla $\mathbf{s}$ pätee $\mathbf{Hs} = \mathbf{0}$. Kun vastaanotettu lohko tulkitaan muodossa $\mathbf{r} = \mathbf{s} + \mathbf{e}$, missä $\mathbf{s}$ on lähetetty koodisana ja $\mathbf{e}$ kanavassa tapahtuneet virheet, nähdään,

että

$$\mathbf{Hr} = \mathbf{H}(\mathbf{s}+\mathbf{e}) = \mathbf{Hs}+\mathbf{He} = \mathbf{He}.$$

Vastaanotettu viesti $\mathbf{r}$ voidaan siten dekoodata yksinkertaisesti laskemalla matriisitulo $\mathbf{Hr}$ ja etsimällä $\mathbf{e}$, jolla $\mathbf{Hr} = \mathbf{He}$. Mahdollisia virhevektoreita $\mathbf{e}$ on olemassa vain $\sum_{i=0}^{m} \binom{n}{i}$ kappaletta, missä $m = \frac{d-1}{2}$ on lohkosta enintään korjattavien virheiden lukumäärä, esimerkiksi Hamming-koodeilla yksi. Kun virhevektori tunnetaan, voidaan lähetetty koodisana laskea helposti $\mathbf{s} = \mathbf{r} - \mathbf{e}$.

Syndroomadekoodaus on nopeaa, mutta sillä voidaan korjata vain koodin etäisyyden d osoittama lukumäärä $\frac{d-1}{2}$ virheitä, joten, luvussa 2 esitettyjen perustelujen nojalla, yksin sitä dekoodaukseen käyttävä algoritmi ei koskaan voi olla kovin tehokas. Esimerkiksi harva parillisuustarkistuskoodi, joka on tehokas ja johon keskitymme vielä myöhemmin, on tyypiltään lineaarinen lohkokoodi, mutta hyödyntää kahdesta käsittelemästämme ominaisuudesta vain koodaamista generaattorimatriisilla. Pelkkää syndroomadekoodausta hyödyntäviä FEC-algoritmeja voidaankin pitää jo vanhanaikaisina.

3.2 Konvoluutiokoodit

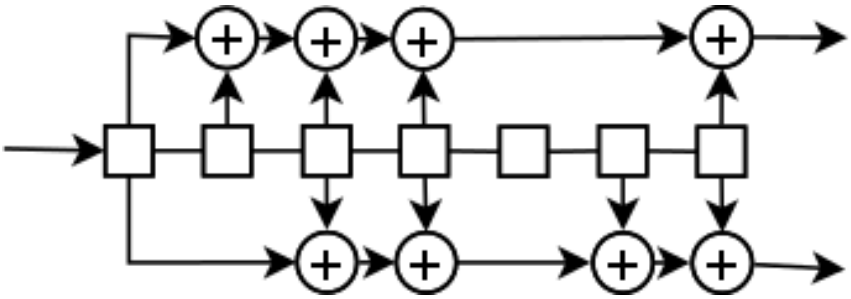
Lohkokoodit jakavat lähetettävät bitit tietynmittaisiin viesteihin, jotka lähetetään kanavaan koodisanoina. Konvoluutiokoodit poikkeavat tästä perustavalla tavalla, sillä ne eivät paloittele lähetettäviä bittejä erillisiin osiin, vaan käsittelevät ne ikään kuin jatkuvana kooderin läpi etenevänä nauhana. Tällä periaate-erolla ei ole käytännön kannalta sikäli merkitystä, että toista voitaisiin käyttää vain joihinkin tiettyihin käytännön sovelluksiin. Sen sijaan se johtaa ominaisuuksiltaan hieman erilaisiin koodeihin, joilla molemmilla voi olla käyttönsä toisistaan hieman erilaisissa kanavissa tai tarkoituksissa.

Tässä alaluvussa tutustumme edellä mainittuun eroon konvoluutiokoodien toimintaperiaatteen kautta, ja toteamme luvussa 2 lohkokoodeilla motivoitujen etäisyyden ja informaatiosuhteen olevan yhtä lailla myös konvoluutiokoodien suorituskyvyn mittareita.

Konvoluutiokoodit koodaavat siis bittivirtaa yksi bitti kerrallaan. Vain tähän yhteen bittiin perustuen ei kuitenkaan voitaisi luoda mitään toistokoodista poikkeavaa algoritmia, joten koodaukseen käytetään myös kooderin tätä tarkoitusta varten *muistirekistereihin* tallettamaa muutamaa edellistä bittiä. Koodauksen aluksi jokaiseen muistirekisteriin on “tallennettu” arvo 0, ja jokaisella algoritmin *askeleella* luettu bitti tallennetaan ensimmäiseen rekisteriin, jossa ollut bit-

ti toiseen rekisteriin ja niin edelleen, kunnes viimeisessä rekisterissä ollutta bittiä ei tallenneta enää mihinkään.

Muistirekisterien määrää merkitään $K-1$, missä konvoluutiokoodin *rajoitteiden määrä* K kertoo suoraan, kuinka moneen bittiin koodaus perustuu. Nämä ovat siis sekä juuri luettu bitti z_0 että muistissa olevat edelliset bitit $z_1, \ldots, z_{k-1}$. Varsinainen koodaus tapahtuu yksinkertaisesti laskemalla näiden yhdestä tai useammasta osajoukosta summat (modulo 2), ja lähettämällä saadut bitit kanavaan ennalta määritetyssä järjestyksessä. Jos osajoukkoja käytetään n kappaletta, kanavaan lähetetään jokaista yksittäistä luettua bittiä kohti n bittiä, joten konvoluutiokoodien informaatiosuhde on $R = \frac{1}{n}$. Jotakin näistä summista voidaan käyttää myös jonkin toisen summan yhtenä yhteenlaskettavana, jolloin koodi on *rekursiivinen*.



Kuva 8: Konvoluutiokoodi rajoitteiden määrällä $K = 7$ ja informaatiosuhteella $R = \frac{1}{2}$.

Esimerkiksi kuvassa 8 esitetyssä, Voyager-luotaimissa käytetyssä ja sen jälkeen suurta suosiota saaneessa konvoluutiokoodissa lähetettävät bitit lasketaan kahdesta osajoukosta, eli informaatiosuhde on $R = \frac{1}{2}$. Ensimmäinen lähetettävä bitti on summa $z_0 + z_1 + z_2 + z_3 + z_6$ ja jälkimmäinen $z_0 + z_2 + z_3 + z_5 + z_6$. Näitä puolestaan kutsutaan konvoluutiokoodin *generaattoripolynomeiksi*, ja niitä voidaan merkitä lyhyemmin `1111001` ja `1011011`, joissa kukin `1` merkitsee vastaavan luetun bitin tai rekisterissä olevan bitin summaan laskemista. Edelleen nämä binääriluvut muutetaan joskus kahdeksanjärjestelmään, jolloin voidaan kirjoittaa esimerkiksi tämän koodin generaattoripolynomien olevan 171 ja 133.

Konvoluutiokoodi voidaan toteuttaa myös siten, että bittejä luetaan useita yhdellä askeleella, ennen lähetettävien bittien laskemista. Tällaisen koodin voidaan ajatella käsittelevän bittivirtaa lohkoissa. Se ei kuitenkaan ole lohkokoodi, siinä mielessä kuin ne määrittelimme, sillä yksittäisen lohkon tuottamat koodatut bitit ovat edelleen riippuvaisia muistirekisterin sisällöstä, siis edellisistä lohkoista. Jos tämän tyyppisessä koodissa kerralla luettavan lohkon pituus on k, on koodin informaatiosuhde $R = \frac{k}{n}$.

Konvoluutiokoodi dekoodataan tyypillisesti *Viterbin algoritmilla*. Tämä on liian laaja tässä työssä kuvattavaksi, mutta siitä on hyvä tietää, että se antaa parhaan mahdollisen dekoodauksen

(suurimman tilastollisen uskottavuuden mielessä), joskin sen laskennallinen vaativuus kasvaa nopeasti konvoluutiokoodin rajoitteiden määrän kasvaessa. Konvoluutiokoodin etäisyys d taas on kahden koodatun sekvenssin pienin ero mitattuna eroavissa biteissä. Se määrittää pienimmän virhemäärän $\frac{d-1}{2}$, jolla dekoodaus vielä onnistuu varmasti. Konvoluutiokoodin rakenteesta johtuen yksittäiset virheet vaikuttavat pääasiassa lähiympäristönsä dekoodaukseen, joten tätä voidaan ennemminkin pitää yksittäisen virhepurskeen suurimpana vielä sallittavana pituutena, kunhan nämä purskeet ovat riittävän kaukana toisistaan. Luvun 2 varoitukset pätevät myös konvoluutiokoodin etäisyyden käyttämiseen sen hyvyyden indikaattorina.

Myös pelkkään konvoluutiokoodaukseen perustuvat FEC-algoritmit ovat käyneet vanhanaikaisiksi, vaikka niitä toki käytetään edelleen vanhoissa järjestelmissä, kuten GSM-tekniikassa. Konvoluutiokoodauksella on silti käyttönsä moderneissakin järjestelmissä, mutta hieman monimutkaisempien ja tehokkaampien algoritmien osina.

3.3 Ketjukoodit

Usean virheenkorjausalgoritmin hyviä puolia voidaan pyrkiä yhdistämään ohjaamalla signaali usean kooderin läpi. Tätä voidaan tarkastella yhtenä suurena algoritmina, jolloin sitä kutsutaan ketjukoodiksi. Useimmiten käytetään kahta virheenkorjausalgoritmia, jolloin koodaamaton signaali menee ensin *ulompaan* ja sen jälkeen *sisempään* kooderiin. Vastaavasti kanavan läpi kulkenut signaali ohjataan ensin *sisempään* ja sen jälkeen *ulompaan* dekooderiin.

Ketjukoodista on erityisesti hyötyä purskevirheiden korjauksessa. Hyviä tuloksia on saavutettu käyttämällä sisempänä koodina konvoluutiokoodia melko vähäisellä rajoitteiden määrällä ja ulompana koodina suuren lohkokoon lineaarista lohkokoodia. Tämä perustuu siihen, että konvoluutiokoodien suorituskyky virheenkorjauksessa on lohkokoodeihin verrattuna suuri, johtuen niiden monimutkaisesta redundanssista jonka Viterbin algoritmi hyödyntää täysin, mutta ne ovat huonoja korjaamaan purskevirheitä. Purskevirheiden korjaus vaatisi konvoluutiokoodilta paljon rajoitteita, mutta niitä ei voida lisätä tarpeeksi dekoodauksen monimutkaistumisen vuoksi. Lohkokoodilta taas vaaditaan samaan suuri lohkokoko, jota nimenomaan voidaan kasvattaa helposti lineaarisuuden takaamien ominaisuuksien ansiosta.

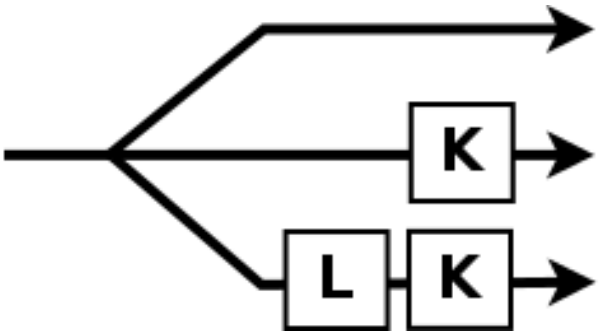
Eräs merkittävä esimerkki edellä kuvatusta on NASAn 1970-luvulla alun perin Voyager-ohjelmaa varten käyttöön ottama ketjukoodi, joka koostuu Reed-Solomon(255,223) -ulkokoodista ja edellisessä alaluvussa kuvatusta, generaattoripolynomien 171 ja 133 määrittämästä sisäkoodista, joiden välissä on vielä lomittelija tuomassa lisäsuojaa purskevirheiltä. Tätä koodia suosittiin

pitkään satelliittikommunikaatiossa, ja se on edelleen käytössä esimerkiksi digitaalitelevision DVB-S -standardissa [1]

Ketjukoodit olivat pitkään suorituskykyisimpiä virheenkorjausalgoritmeja ja, vaikka esimerkiksi edellä kuvattu Voyager-ohjelman koodi on nykyisin vanhanaikainen, ajatus kahden suorituskykyisen koodin hyvien puolien yhdistämisestä on edelleen käytännöllinen. Esimerkiksi DVB-S -standardin jo osin korvanneeseen seuraajaan sisä- ja ulkokoodit on vain vaihdettu nykyaikaisempiin [2].

3.4 Turbokoodit

Turbokoodit ovat nykyaikaisia, erittäin suorituskykyisiä virheenkorjausalgoritmeja. Niitä on useita erilaisia, mutta pelkistetyimmillään ne koostuvat kahdesta konvoluutiokoodista ja lomittelijasta, jotka on kooderissa kytketty kuvan 9 mukaisesti. Tämä kooderi lähettää databitit sellaisenaan, lisäten niihin kaksi sarjaa tarkistusbittejä. Ensimmäinen sarja on yksinkertaisesti alkuperäinen data konvoluutiokoodilla koodattuna, toinen sama data lomiteltuna ja sen jälkeen konvoluutiokoodilla koodattuna. Tällä saadaan aikaan hyvin monimutkainen, useiden virheiden korjaamisen mahdollistava redundanssi.



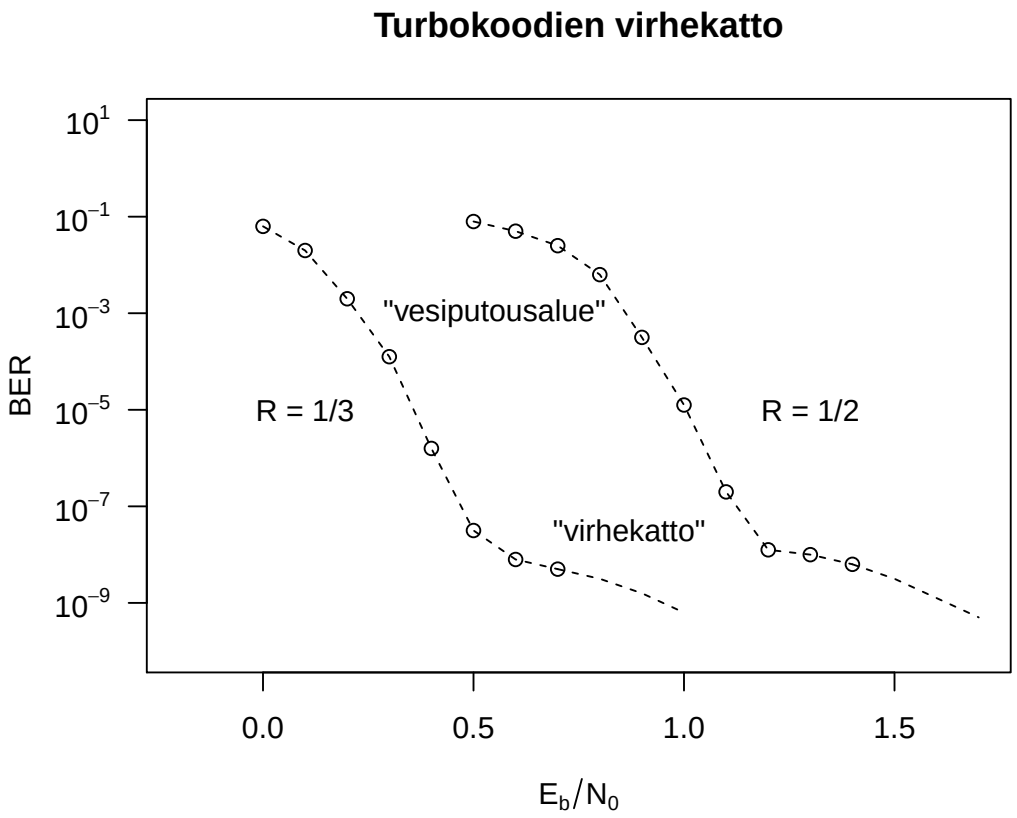
Kuva 9: Turbokoodin muodostuminen kahdesta konvoluutiokoodista ja lomittelijasta. Lähetettävään viestiin lisätään tässä esimerkissä kutakin bittiä kohden kaksi tarkistusbittiä.

Monimutkaisia koodeja toki pystytään rakentamaan muillakin tavoin, mutta olennaista on, pystytäänkö ne dekoodaamaan järkevässä ajassa. Kuten aiemmin tässä luvussa käsiteltiin, lineaariset lohkokoodit dekoodataan tavallisesti syndroomadekoodauksella, joka on nopea mutta täysin riippuvainen koodin etäisyydestä, ja konvoluutiokoodit tavallisesti Viterbin algoritmilla, joka päinvastoin ei ole etäisyyden rajoittama, mutta liian hidas monimutkaisiin koodeihin.

Turbokoodin dekooderi koostuu kahdesta konvoluutiokoodin dekooderista, lomittelijaan yhdistettynä, jotka arvioivat todennäköisyyksiä sille, että mikäkin vastaanotettu bitti on `1` tai `0`. Dekooderi ilman lomittelijaa arvioi näitä todennäköisyyksiä lomittelemattomien tarkistusbittien

perusteella ja päinvastoin. Kun molemmat niistä ovat ratkaisseet dekoodauksen osaltaan, ne vaihtavat arvioitaan kunkin bitin todennäköisestä arvosta ja jatkavat tätä, kunnes pääsevät yksimielisyyteen. Tämänkaltainen dekooderi pystyy ratkaisemaan paljon enemmän kuin vain koodin etäisyyden osoittaman määrän virheitä, mikä on hyvän koodin edellytys, kuten luvussa 2 totesimme, eikä se siltikään ole laskennallisuudeltaan liian vaativa.

Huonona puolena turbokoodit kärsivät *virhekatosta*, mikä tarkoittaa että kanavaa parannettaessa niiden suorituskyky kasvaa aluksi odotetussa suhteessa, mutta jossakin vaiheessa tämä kasvu tasaantuu nopeasti lähes olemattomaksi. Virhekattoa on havainnollistettu kuvassa 10. Tavanomaisilla lohkokoodeilla ja konvoluutiokoodeilla tällaista ei esiinny, vaan niiden suorituskyvyn kasvu jatkuu tasaisesti kanavaa parannettaessa. Vaikka tämän virhekaton alentamiseen onkin useita tekniikoita, turbokoodit eivät välttämättä ole sellaisenaan hyviä ratkaisuja tässä mielessä liian hyvälaatuisiin kanaviin. Tästä huolimatta turbokoodit ovat ansainneet viime vuosikymmeninä paikkansa lukuisissa sovelluksissa, kuten 3G-tekniikassa, ja ne ovat harvojen parillisuustarkistuskoodien rinnalla nykyisin tehokkaimmat ratkaisut virheettömän viestinnän tavoitteluun.



Kuva 10: Kahden turbokoodin simuloitu suorituskyky. Saavutettu bittivirhesuhde ei kehity kuvaajien alkupään mukaisesti, vaan tasaantuu jossakin vaiheessa. Kuvassa esitetyt tulokset perustuvat raportin [3] simulaatioihin.

3.5 Harvat parillisuustarkistuskoodit

Harvat parillisuustarkistuskoodit (engl. Low-Density Parity-Check, LDPC) ovat lineaarisia lohkokoodeja, jotka ovat paitsi määritelmän 2.12 mukaan hyviä, joiden etäisyys on myös määritelmän 2.8 mukaan hyvä. Kuten turbokoodien, myös LDPC-koodien tehokkuus perustuu iteratiiviseen todennäköisyyksien laskentaan dekoodauksessa, joka mahdollistaa useampien virheiden korjaamisen kuin pelkkä etäisyysominaisuus antaisi olettaa.

LDPC-kooderi toimii aivan kuten tavallinen lineaarisen lohkokoodin kooderi, laskemalla matriisitulon lähetettävien bittien ja koodin generaattorimatriisin välillä. Dekoodaus tapahtuu *uskottavuuspropagointimenetelmällä*, jossa vastaanotetut bitit jaetaan viestibitteihin ja parillisuustarkistusbitteihin, joista parillisuustarkistusbitit sitten välittävät niihin liittyville biteille tietoa arvosta, joka niillä pitäisi olla muut bitit huomioiden, ja viestibitit päivittävät arvoaan viestien siitä parillisuustarkistusbiteille. Tämä tehokas menetelmä tunnetaan myös nimillä *summa-tulo -algoritmi* ja *viestinvälitysalgoritmi*.

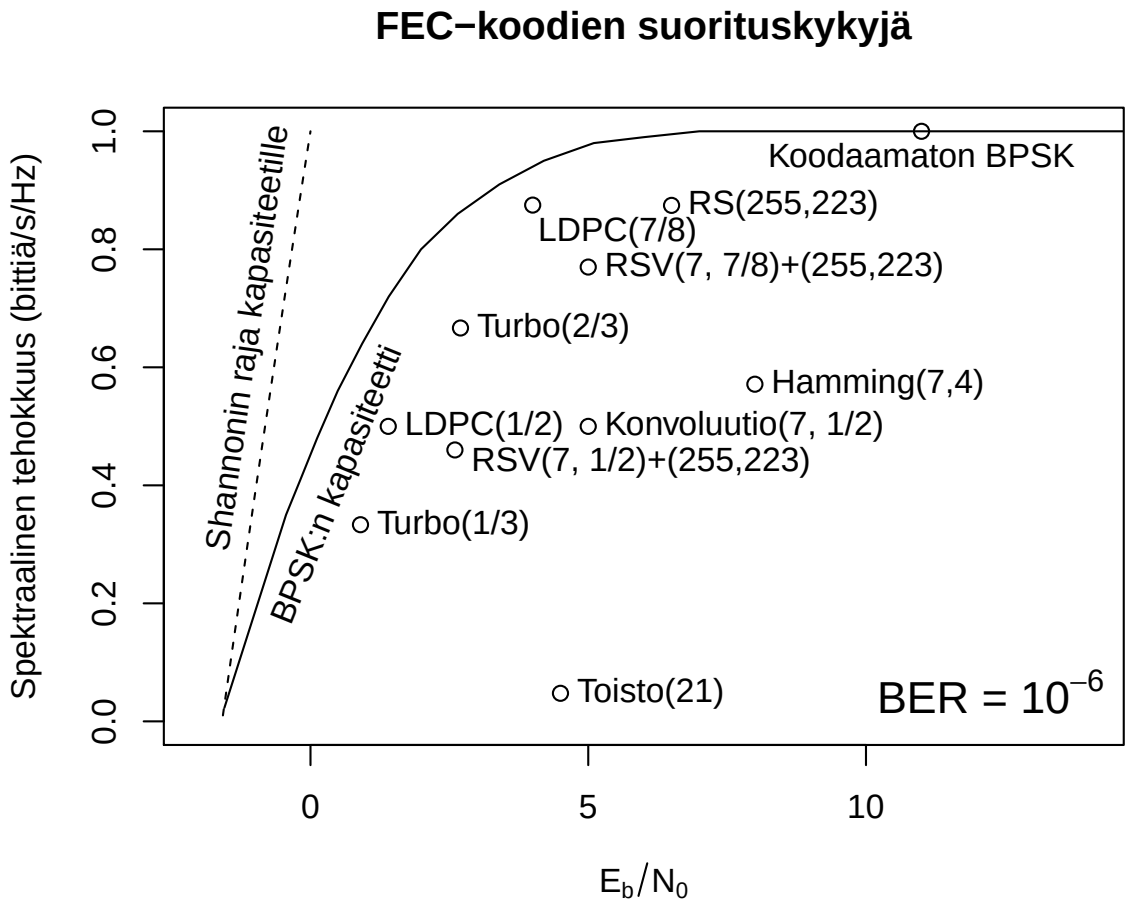
Uskottavuuspropagointimenetelmän tehokkuuden takaamiseksi generaattorimatriisin $\mathbf{G} = [\mathbf{I}_k|\mathbf{P}]$ on oltava sellainen, että parillisuustarkistusmatriisi $\mathbf{H} = [\mathbf{P}^T|\mathbf{I}_{n-k}]$ on *harva*, tarkoittaen pientä ykkösten määrää jokaisessa rivissä ja sarakkeessa, mistä koko algoritmi on saanut nimensä. Tämä mahdollistaa myös koodauksessa käytettävien matriisitulojen laskennan sekä itse dekoodauksen optimointia [20, 18].

LDPC-koodit ovat turbokoodien ohella ainoat tunnetut virheenkorjausalgoritmit, joilla on niin teoriassa kuin käytännössäkin päästy hyvin lähelle luvussa 2 esiteltyä rajaa virheenkorjauksen suorituskyvylle. Ne molemmat tosin kärsivät edellisessä alaluvussa kuvatusta virhekattoilmiöstä. Koska LDPC-koodit ovat viime vuosina jättäneet taakseen turbokoodit monissa kilpailuissa paikoista standardeissa [2, 17, 4], voitaneen niitä pitää jopa tämän hetken parhaina ratkaisuina virheenkorjaukseen.

3.6 Algoritmien vertailu

Kuvaan 11 on koottu esimerkit kaikista tässä työssä käsitellyistä virheenkorjausalgoritmeista. Vertailu on tehty käyttäen menetelmää joka luvussa 2 todettiin soveltuvan parhaiten erilaisten informaatiosuhteiden koodien vertailuun, eli piirtämällä koodit niiden spektraalisen tehokkuuden ja kohinan spektraaliteholla jaetun bittikohtaisen energian suhteen. Kuten kuvas-

ta nähdään, nykyaikaiset virheenkorjauskoodit ovat suorituskyvyltään jo hyvin lähellä lauseen 2.10 määräämää ylärajaa. Kaikkein lähimpänä ovat turbo- ja LDPC-koodit, mutta myös Reed-Solomon/Viterbi -ketjukoodilla voidaan päästä melko lähelle, ainakin verrattuna yksinkertaisempiin koodeihin.



Kuva 11: Kanavakoodauksien spektraalisia tehokkuuksia kohinan spektraalitiheydellä jaetun bittikohtaisen energian funktiona, kun bittivirhesuhde on kiinnitetty arvoon 10^{-6} ja modulaatiomenetelmä on BPSK. Konvoluutiokoodi vastaa luvussa aiemmin esiteltyä esimerkkiä rajoitteiden määrällä 7 ja informaatiosuhteella 1/2. RS tarkoittaa Reed-Solomon -lohkokoodia, kun taas RSV on esimerkki ketjukoodista, jossa ulkokoodina on Reed-Solomon ja sisäkoodi Viterbi-dekoodattu konvoluutiokoodi. RSV-, Turbo- ja LDPC -koodeista kuvaan on valittu jokaisesta kaksi eri informaatiosuhteen esimerkkiä. Kuvassa esitetyt tulokset perustuvat lähteiden [3, 24] simulaatioihin.

4 Johtopäätökset

FEC-algoritmien suorituskykyä voi mitata ainakin koodin etäisyydellä, hyvyydellä suhteessa kanavan kapasiteettiin sekä koodausvahvistuksella. Koodausvahvistus on näistä konkreettisin, joskin sen laskemiseksi on oletettava häiriötaso, jota taas todellisuudessa ei voida valita, eikä se myöskään huomioi koodin informaatiosuhdetta. Suositeltavimpana tapana pidän koodausvahvistusten vertailemista spektraaliseen tehokkuuteen. Mikäli halutaan tutkia esimerkiksi virhekattoa, tulee kuitenkin tutkia koodausvahvistusta E_b/N_0-suhteen kuvaajana, pitäen mielessä edelliset huomiot.

Joissakin sähköisen kommunikaation yleisteoksissa, kuten esimerkiksi [10, 14], otetaan esille vain koodin etäisyys mittarina sen hyvyydelle. Tässä lienee kyseessä osittain pedagoginen valinta, sillä koodin etäisyys saattaa olla helppo ymmärtää. Pidän etäisyyttä kuitenkin huonona mittarina, sillä huononkin etäisyyden koodi saattaa itse asiassa olla suorituskyvyltään hyvä [15]. Viimeaikaisessa tutkimuskirjallisuudessa koodien hyvyyttä ei vaikuteta arvioitavan niiden etäisyyden perusteella, joten alan asiantuntijoiden käytännöissä en näe moitittavaa.

Tässä työssä virheenkorjauskoodien suorituskykyä tarkasteltiin virhe- ja informaatiosuhteiden näkökulmasta, mutta käytännössä niiden valintaan vaikuttavat lisäksi luonnollisesti ainakin tarvittavan laitteiston monimutkaisuus ja hinta, koodausmenetelmän aiheuttaman viiveen suuruus sekä kanavan rajoitteet teholle ja kaistanleveydelle.

FEC-algoritmit jaetaan tavallisesti lohko- ja konvoluutiokoodeihin, joskin tarkasti ottaen jälkimmäiset voi nähdä myös edellisten erikoistapauksena. Sovelluksissa merkittävin ero näiden välillä on siinä, että lohkokoodit saattavat sietää purskevirheitä konvoluutiokoodeja paremmin, sillä lohkokoodien lohkokokoa voidaan kasvattaa melko suureksikin, mutta konvoluutiokoodin rajoitteiden määrä on pidettävä pienenä dekoodauksen mahdollistamiseksi käytännöllisessä ajassa. Purskevirheet voidaan kuitenkin huomioida lomittelulla.

Tällä hetkellä suorituskykyisimpiä algoritmeja ovat konvoluutiokoodeihin kuuluvat turbokoodit, lohkokoodeihin kuuluvat LDPC-koodit sekä ketjukoodit, joissa käytetään ulompana koodina pitkän lohkokoon lohkokoodia ja sisempänä jotakin modernia konvoluutiokoodia. Käyttöön ovat tulleet myös adaptiiviset tekniikat, joissa käytettyä koodia mukautetaan vastaamaan kanavan häiriötasoa [21]. Tällainen on käytössä esimerkiksi HSDPA ja WiMAX -standardeissa [16].

Viitteet

[1] EN 300 744 V1.5.1 Digital Video Broadcasting (DVB); Framing structure, channel coding and modulation for digital terrestrial television. European Telecommunications Standards Institute, 2004. 64 s.

[2] EN 302 307-1 V1.4.1 Digital Video Broadcasting (DVB); Second generation framing structure, channel coding and modulation systems for Broadcasting, Interactive Services, News Gathering and other broadband satellite applications; Part 1: DVB-S2. European Telecommunications Standards Institute, 2014. 80 s.

[3] *TM Synchronization and Channel Coding – Summary of Concept and Rationale*. Tekninen raportti CCSDS 130.1-G-2, The Consultative Committee for Space Data Systems, 2012. [viitattu 27.3.2015]. Saatavissa: http://public.ccsds.org/publications/archive/130x1g2.pdf.

[4] K. Andrews, D. Divsalar, S. Dolinar, J. Hamkins, C. Jones, ja F. Pollara. *The Development of Turbo and LDPC Codes for Deep-Space Applications*. Proceedings of the IEEE, 95(11):2142–2156, 2007.

[5] S-Y Chung, T. Richardson, ja R. Urbanke. *Analysis of sum-product decoding of low-density parity-check codes using a Gaussian approximation*. IEEE Transactions on Information Theory, 47(2):657–670, 2006.

[6] E. Elliott. *Estimates of Error Rates for Codes on Burst-Noise Channels*. Bell System Technical Journal, 42:1977–1997, 1963.

[7] R. Gallager. *Low-Density Parity-Check Codes*. M.I.T. Press, 1963.

[8] H. El Gamal ja A. Hammons. *Analyzing the turbo decoder using the Gaussian approximation*. IEEE Transactions on Information Theory, 47(2):671–686, 2006.

[9] E. Gilbert. *Capacity of a burst-noise channel*. Bell System Technical Journal, 39:1253–1265, 1960.

[10] I. Glover ja P. Grant. *Digital Communications*, sarjassa *Pearson education*. Pearson education. Pearson/Prentice Hall, 2004.

[11] A. Goldsmith. *Wireless Communications*. Cambridge University Press, New York, NY, USA, 2005.

[12] R. Hamming. *Error Detecting and Error Correcting Codes*. The Bell System Technical Journal, 26(2):147–160, 1950.

[13] J. Kosola ja T. Solante. *Digitaalinen taistelukenttä*. Maanpuolustuskorkeakoulu, sotatekniikan laitos, julkaisusarja 1, 2013.

[14] B. Lathi ja Z. Ding. *Modern Digital and Analog Communications Systems International*, sarjassa *Oxford series in electrical and computer engineering*. Oxford series in electrical and computer engineering. Oxford University Press, Incorporated, 2010.

[15] D. MacKay. *Information Theory, Inference & Learning Algorithms*. Cambridge University Press, New York, NY, USA, 2002.

[16] C. Mehlführer. *Measurement-based Performance Evaluation of WiMAX and HSDPA*. Väitöskirja, Technische Universität Wien, 2009.

[17] X. Peng ja S. Goto. *Implementation of LDPC decoder for 802.16e*. Kirjassa ASIC, 2009. ASICON '09. IEEE 8th International Conference on, ss. 501–504, 2009.

[18] P. Radosavljevic, A. De Baynast, ja J. Cavallaro. *Optimized Message Passing Schedules for LDPC Decoding*. Kirjassa Conference Record of the Thirty-Ninth Asilomar Conference on Signals, Systems and Computers, 2005, ss. 591–595, 2005.

[19] I. Reed ja G. Solomon. *Polynomial Codes Over Certain Finite Fields*. Journal of the Society for Industrial and Applied Mathematics, 8(2):300–304, 1960.

[20] T. Richardson ja R. Urbanke. *Efficient Encoding of Low-Density Parity-Check Codes*. IEEE Transactions on Information Theory, 47(2):638–656, 2001.

[21] P. Rysavy. *Challenges and Considerations in Defining Spectrum Efficiency*. Proceedings of the IEEE, 102(3):386–392, 2014.

[22] A. Scardicchio, F. Stillinger, ja S. Torquato. *Estimates of the optimal density of sphere packings in high dimensions*. Journal of Mathematical Physics, 49(4):043301, 2008.

[23] R. Singleton. *Maximum distance q-nary codes*. IEEE Transactions on Information Theory, 10(2):116–118, 1964.

[24] D. Sklar ja C. Wang. *Performance evaluation of two rate 2/3 turbo code configurations*. Kirjassa Aerospace Conference Proceedings, 2002. IEEE, osa 3, ss. 3–1349–3–1354 vol.3, 2002.

[25] R. Yates. *Derivation of the Shannon Spectral Efficiency Limit*. Digital Signal Labs, 2014. [viitattu 27.3.2015]. Saatavissa: http://www.digitalsignallabs.com/shannonlimit.pdf.